CHE221

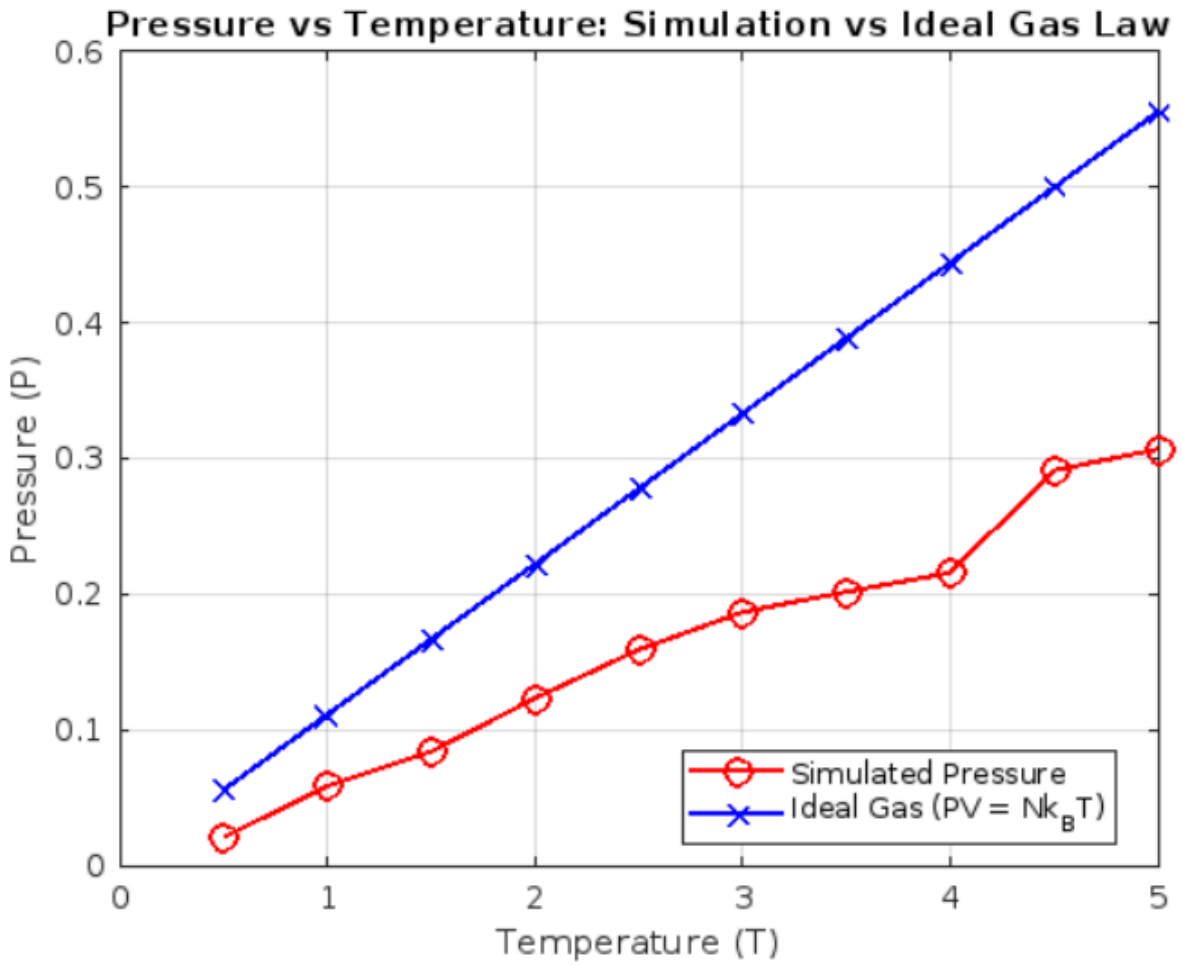
SIMULATION LAB 8

-PRAGATI PATEL (230765) UG CHE

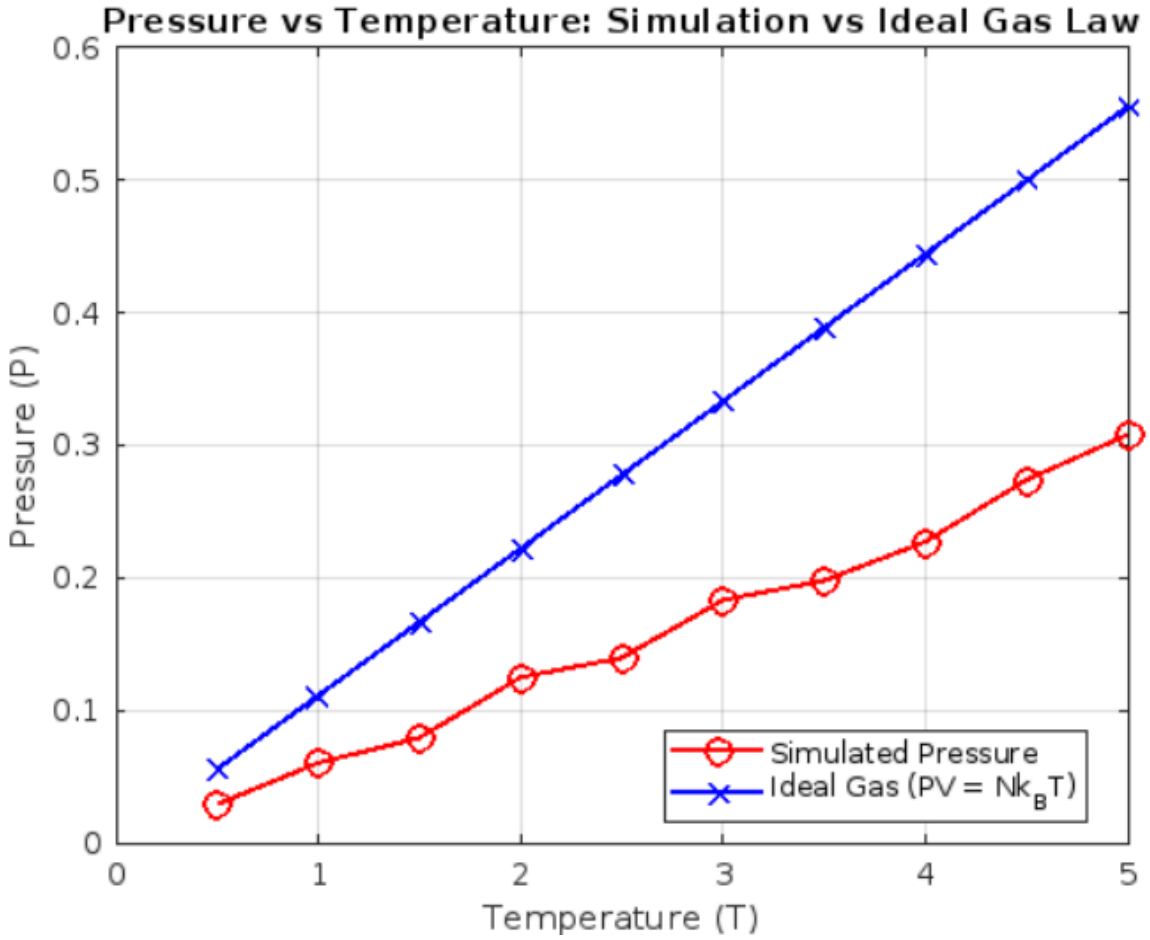
1) Increase the number of steps and comment on how pressure changes.

N=3;

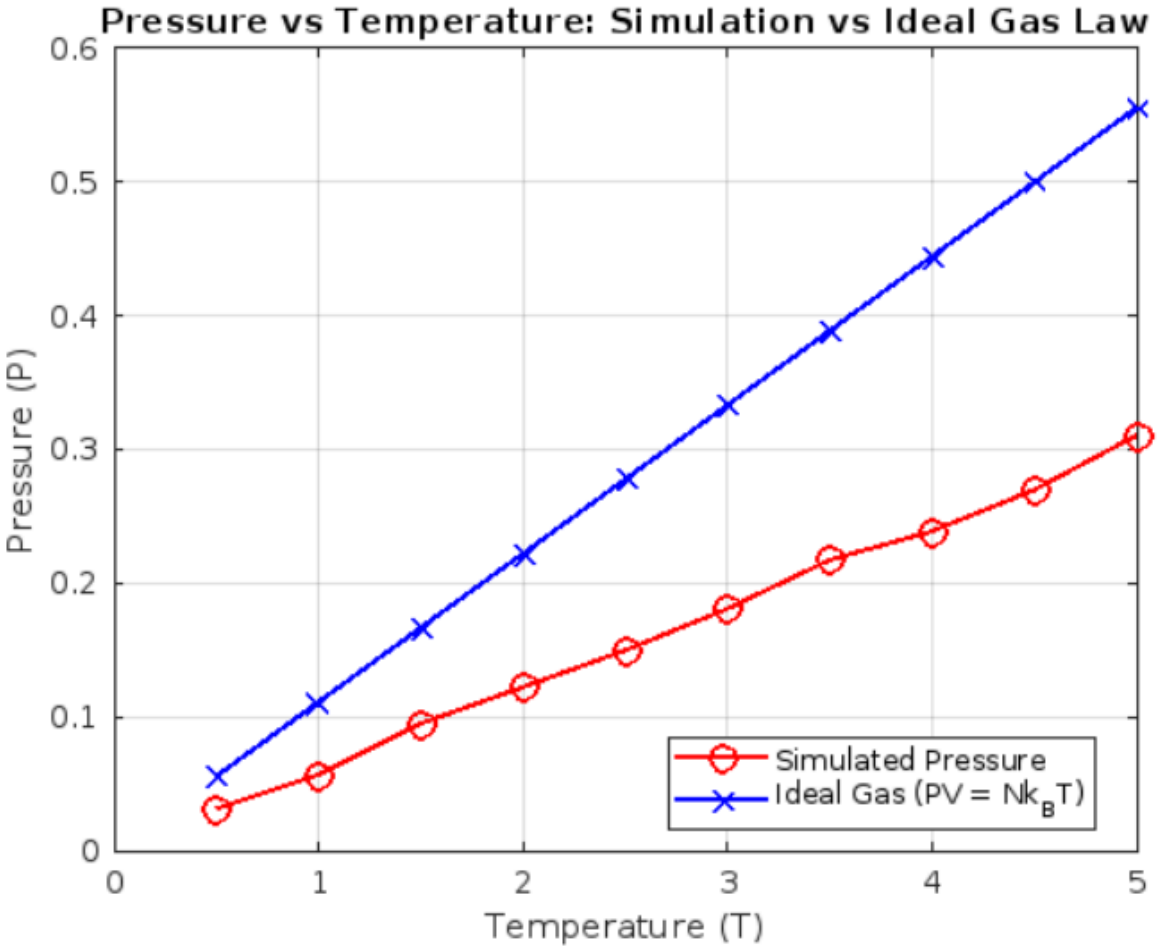
n_steps=number of steps



n_steps=300;


Pressure vs Temperature: Simulation vs Ideal Gas Law
Pressure (P)
Temperature (T)
0
0.1
0.2
0.3
0.4
0.5
0.6
0
1
2
3
4
5
Simulated Pressure
Ideal Gas (PV = NkBT)

n_steps=500;



n_steps=1000;

As the number of simulation steps increases, the pressure variation with temperature tends to become linear due to the system reaching a more statistically stable state. Initially, with fewer steps, fluctuations in molecular interactions and wall collisions cause pressure variations. However, as the simulation progresses, more molecular collisions occur. Since pressure is directly proportional to temperature at a constant volume, extended simulations allow pressure to converge to its expected value, smoothing out statistical fluctuations and making the trend more linear.

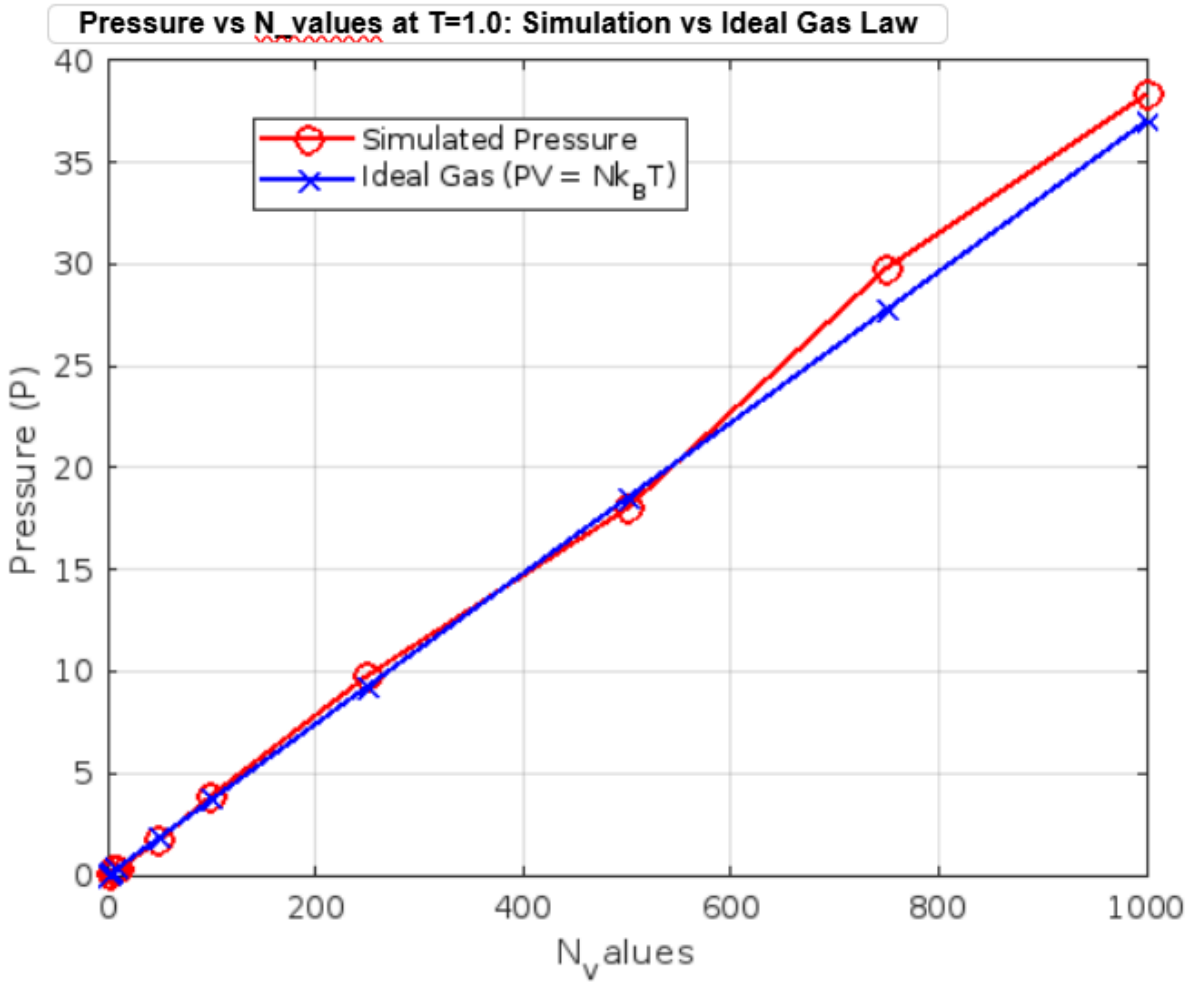
2)Plot pressure as a function of number of particles at T= 1.0 and 5.0. Comment on your observations.

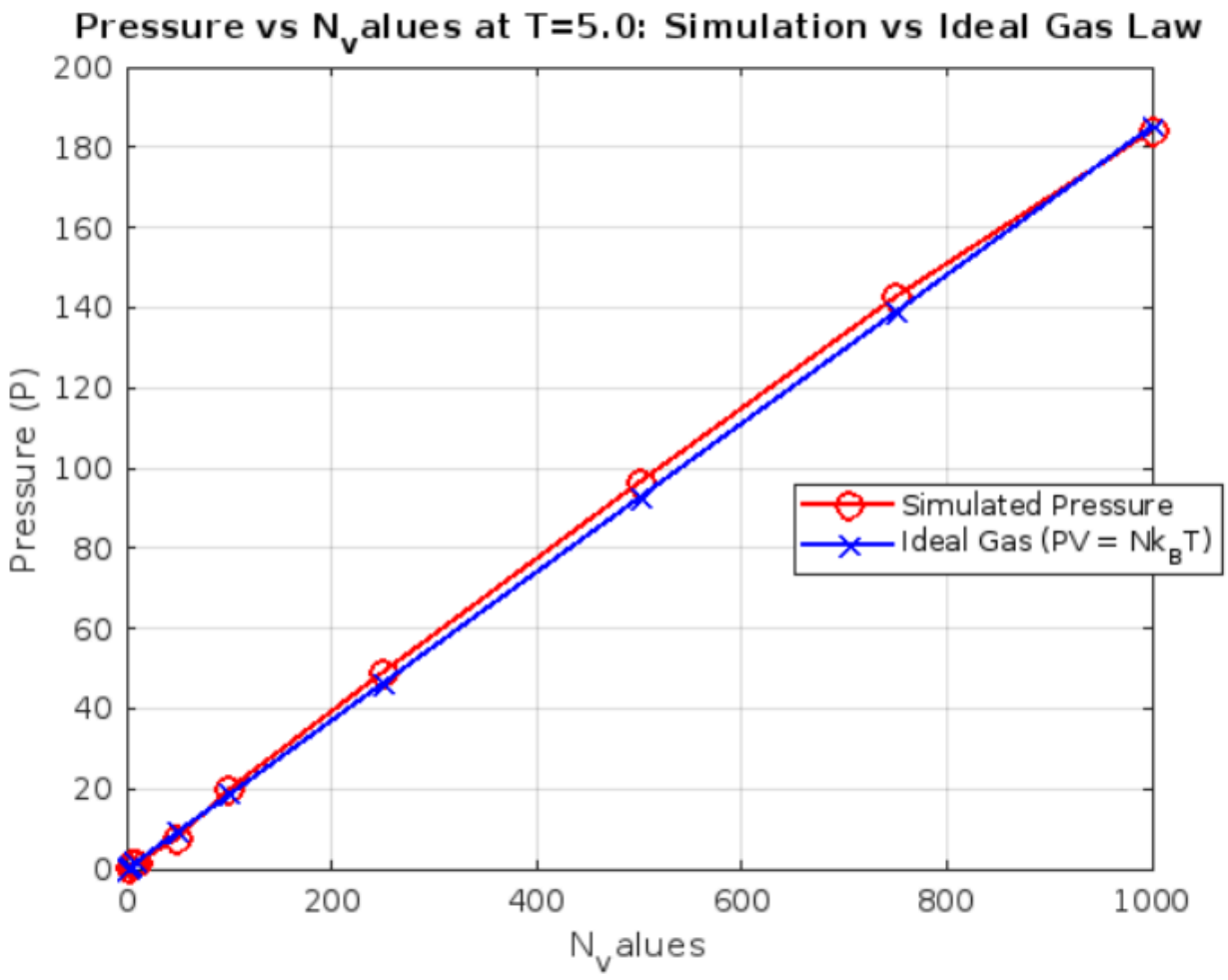
n_steps=100;

N_values=[3,5,10, 50, 100,250, 500,750, 1000];

For T=1.0 there is variation between simulated pressure and ideal gas pressure but at T=5.0 there is minimal variation between simulated pressure and ideal gas pressure.

The simulated pressure matches the ideal gas law more accurately at T=5.0 because the system's thermal energy dominates statistical and numerical fluctuations.
At T=1.0, the effect of finite particle numbers and simulation artifacts leads to a slight overestimation of pressure for larger N.
Pressure at each N_values is 5 times higher than T=1.0 for T=5.0

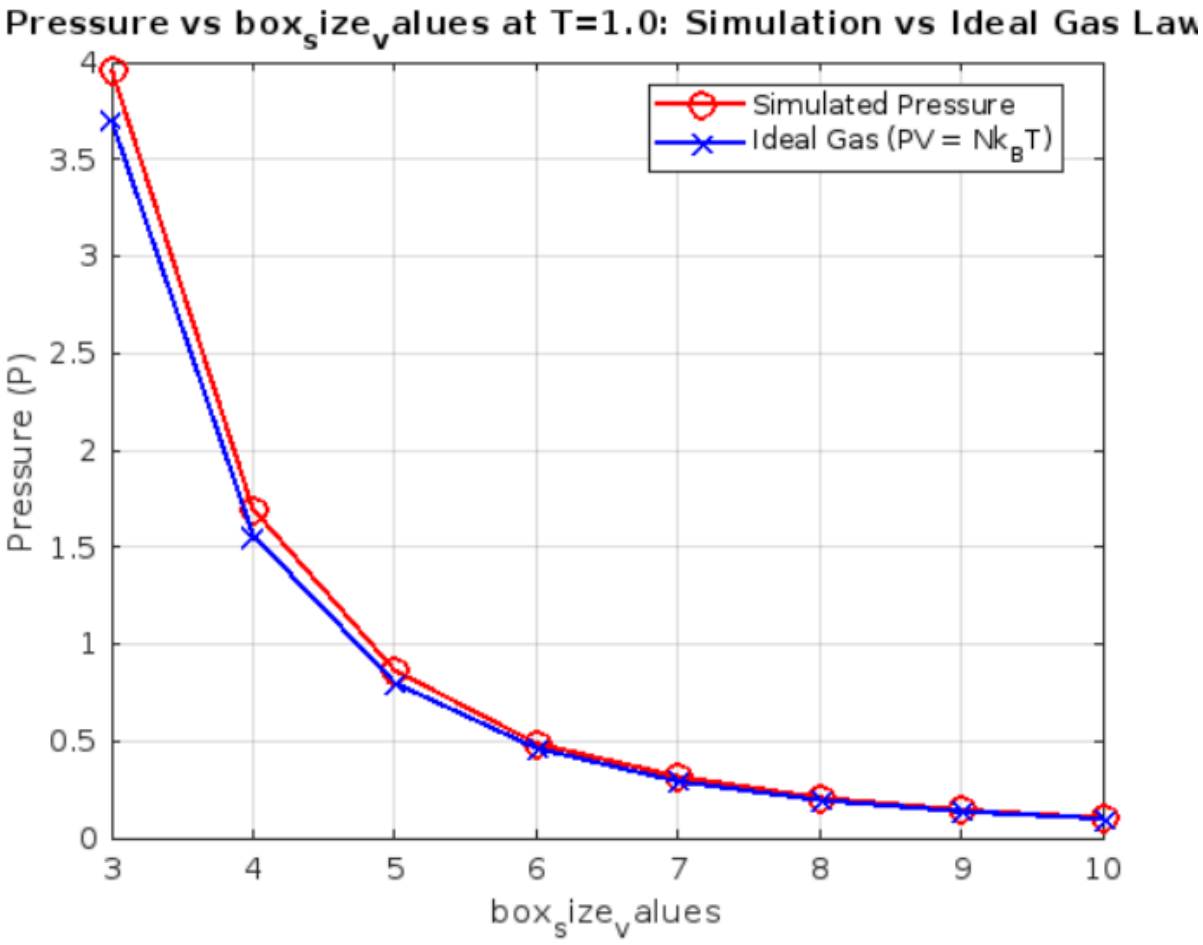




3)Plot pressure as a function of box size at a fixed Temperature (T = 1.0). Comment on your observations.

The two curves are almost identical, confirming that the molecular dynamics simulation correctly models an ideal gas.At large box sizes, the pressure approaches zero, as expected from the ideal gas law.Box size is directly proportional to volume which is inversely proportional to pressure.So pressure decreases at larger box sizes.This is because particle-wall collisions become less frequent, reducing the total momentum transfer to the walls.

```
N = 3;
box_size_values = linspace(3,10,10);
n_steps=1000;
```

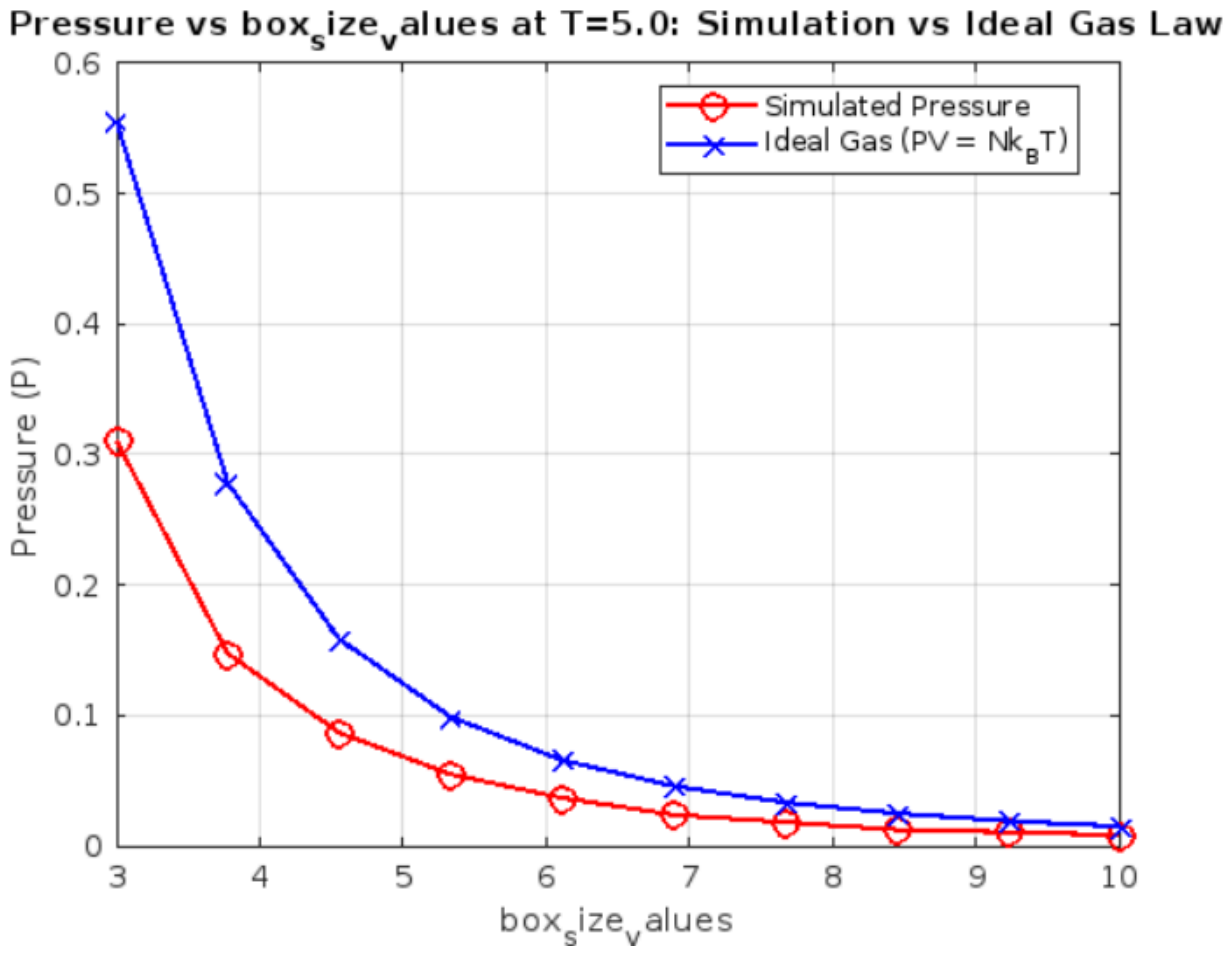


4) How does the pressure change as you increase the box size at a higher temperature (T = 5.0)?

The pressure decreases as box size increases, following the ideal gas law.Since volume increases as, it is directly proportional to cube of box_size, pressure decreases inversely with volume.

Comparing with the T = 1.0 case, the pressure at each box size is approximately 5 times lower.This is expected since the ideal gas law states that pressure is inversely proportional to volume.

```
N = 3;
box_size_values = linspace(3,10,10);
n_steps=1000;
```



APPENDIX:

1)

```
function ideal_gas_sim
   %  Code to simulate MD for ideal gas molecules. Note the following:
   % - N non-interacting particles in a 3D cubic box.
   % - Collisions only with walls (particles bounce elastically).
   % - Pressure from total momentum transfer to walls.
   % - Compare simulated P vs. T to ideal gas law: P = (N/V) * kB * T.
   clear; clc; close all;
   %% Parameters
   N = 3;                  % num of particles
   box_size = 3.0;           % length of cubic box
   mass = 1.0;               % Particle mass
   kB = 1.0;                 % Boltzmann constant ( we have set it to 1 here
for convenience)
   dt = 0.01;                % time step of the simulation
   n_steps = 500;           % num. of simulation steps
```

```
    target_temperatures = linspace(0.5, 5.0, 10);  % Temp. range
    wall_area = 6 * box_size^2;  % Total surface area of the cubic box
    %% Allocate array to store pressure
    pressures = zeros(size(target_temperatures));
    %% Loop over each temperature to perform MD at each temp
    for t_idx = 1:length(target_temperatures)
        T = target_temperatures(t_idx);
        % run MD at this temp.
        [pressures(t_idx), positions_history] = simulate_ideal_gas( ...
            T, N, box_size, mass, n_steps, dt, kB, wall_area);
        % show a visualization of the MD for the last temp.
        if t_idx == length(target_temperatures)
            figure('Name','Particle Animation','Color','w');
            for step_i = 1:size(positions_history, 1)
                scatter3( positions_history(step_i, :, 1), ...
                          positions_history(step_i, :, 2), ...
                          positions_history(step_i, :, 3), ...
                          15, 'filled' );
                axis([0 box_size 0 box_size 0 box_size]); axis square;
                xlabel('X'); ylabel('Y'); zlabel('Z');
                title(sprintf('Positions at time step = %d', step_i*10));
                drawnow;
                pause(0.02);
            end
        end
    end
    %% Compare P-T data results from the sim w/ the ideal gas law: P = (N * kB
* T) / V
    % (kB=1 and V=box_size^3, therefore P_ideal = (N * T) / (box_size^3))
    volume = box_size^3;
    P_ideal = (N .* target_temperatures) / volume;
    %% Plot results
    figure('Name','Pressure vs Temperature','Color','w');
    plot(target_temperatures, pressures, 'ro-','LineWidth',1.5,'MarkerSize',8);
    hold on;
    plot(target_temperatures, P_ideal, 'bx-','LineWidth',1.5,'MarkerSize',8);
    xlabel('Temperature (T)');
    ylabel('Pressure (P)');
    title('Pressure vs Temperature: Simulation vs Ideal Gas Law');
    legend('Simulated Pressure','Ideal Gas (PV = Nk_BT)','Location','best');
    grid on;
end
function [pressure, positions_history] = simulate_ideal_gas( ...
                T, N, box_size, mass, n_steps, dt, kB, wall_area)
    % IDEAL GAS SIMULATION
    % -----------------------------------------------------
    % This function simulates an ideal gas in a box for n_steps at temperature
T
    % Returns:
```

```
%   pressure
%   positions_history

% fix a random seed reproducibility
rng(1);
% initialize particle positions uniformly in the box
positions = box_size * rand(N, 3);
% initialize velocities from the Maxwell-Boltzmann distribution
% YOUR CODE HERE #1

velocities=sqrt((kB*T)/mass).*randn(N,3);
% store positions every 10 steps. We will use this for visualization
save_interval = 10;
n_save = floor(n_steps / save_interval);
positions_history = zeros(n_save, N, 3);
save_counter = 1;

% store total momentum transfer
total_momentum_transfer = 0;
% Simulation loop
for step = 1:n_steps
    % update positions
    % YOUR CODE HERE #2
    positions=positions+velocities*dt;

    % check particle collisions with the walls in each dimension
    for dim = 1:3
        % gas particles hitting the left wall at x=0 (or y=0, z=0)
        mask_left = (positions(:, dim) < 0);
        % reflect position if the particle has gone beyond the boundary
        positions(mask_left, dim) = -positions(mask_left, dim);
        % momentum transfer to the wall: 2*m*|v|
        total_momentum_transfer = total_momentum_transfer ...
            + 2 * mass * sum(abs(velocities(mask_left, dim)));
        % flip velocity after the particle is reflected
        velocities(mask_left, dim) = -velocities(mask_left, dim);
        % gas particles hitting the "right" wall at x=box_size (or
y=box_size, z=box_size)
        % YOUR CODE HERE #3
        mask_right = (positions(:, dim) > box_size);
        % reflect position
        % YOUR CODE HERE #4
        positions(mask_right, dim) = 2*box_size-positions(mask_right, dim);

        % momentum transfer
        % YOUR CODE HERE #5
        total_momentum_transfer = total_momentum_transfer ...
            + 2 * mass * sum(abs(velocities(mask_right, dim)));
```

```
            % flip velocity
            % YOUR CODE HERE #6
           velocities(mask_right, dim) = -velocities(mask_right, dim);
        end
        % save positions periodically
        if mod(step, save_interval) == 0
            positions_history(save_counter, :, :) = positions;
            save_counter = save_counter + 1;
        end
    end
    % calculate final pressure
    %    P = (Total momentum transfer) / (Total time * Wall area)
    simulation_time = n_steps * dt;
    pressure = total_momentum_transfer / (simulation_time * wall_area);
end
```

2)

```
function ideal_gas_sim
    %  Code to simulate MD for ideal gas molecules. Note the following:
    % - N non-interacting particles in a 3D cubic box.
    % - Collisions only with walls (particles bounce elastically).
    % - Pressure from total momentum transfer to walls.
    % - Compare simulated P vs. T to ideal gas law: P = (N/V) * kB * T.
    clear; clc; close all;
    %% Parameters
    N_values =  [3,5,10, 50, 100,250, 500,750, 1000];              % num of
particles
    box_size = 3.0;          % length of cubic box
    mass = 1.0;              % Particle mass
    kB = 1.0;                % Boltzmann constant ( we have set it to 1 here
for convenience)
    dt = 0.01;               % time step of the simulation
    n_steps = 50;          % num. of simulation steps
    target_temperatures =1.0;  % Temp. range
    wall_area = 6 * box_size^2;  % Total surface area of the cubic box
    %% Allocate array to store pressure
     pressures = zeros(size(N_values));
    %% Loop over each temperature to perform MD at each temp
    for t_idx = 1:length(N_values)
        T = target_temperatures;
        N = N_values(t_idx);
        % run MD at this temp.
        [pressures(t_idx), positions_history] = simulate_ideal_gas( ...
            T, N, box_size, mass, n_steps, dt, kB, wall_area);
        % show a visualization of the MD for the last temp.
        if t_idx == length(target_temperatures)
            figure('Name','Particle Animation','Color','w');
            for step_i = 1:size(positions_history, 1)
                scatter3( positions_history(step_i, :, 1), ...
```

```
                    positions_history(step_i, :, 2), ...
                    positions_history(step_i, :, 3), ...
                    15, 'filled' );
            axis([0 box_size 0 box_size 0 box_size]); axis square;
            xlabel('X'); ylabel('Y'); zlabel('Z');
            title(sprintf('Positions at time step = %d', step_i*10));
            drawnow;
            pause(0.02);
        end
    end
end
%% Compare P-T data results from the sim w/ the ideal gas law: P = (N * kB
* T) / V
% (kB=1 and V=box_size^3, therefore P_ideal = (N * T) / (box_size^3))
volume = box_size^3;
P_ideal = (N_values .* target_temperatures) / volume;
%% Plot results
figure('Name','Pressure vs N_values','Color','w');
plot(N_values, pressures, 'ro-','LineWidth',1.5,'MarkerSize',8);
hold on;
plot(N_values, P_ideal, 'bx-','LineWidth',1.5,'MarkerSize',8);
xlabel('N_values');
ylabel('Pressure (P)');
title('Pressure vs N_values at T=1.0: Simulation vs Ideal Gas Law');
legend('Simulated Pressure','Ideal Gas (PV = Nk_BT)','Location','best');
grid on;
end
function [pressure, positions_history] = simulate_ideal_gas( ...
            T, N, box_size, mass, n_steps, dt, kB, wall_area)
% IDEAL GAS SIMULATION
% ---------------------------------------------------------
% This function simulates an ideal gas in a box for n_steps at temperature
T
% Returns:
%   pressure
%   positions_history

% fix a random seed reproducibility
rng(1);
% initialize particle positions uniformly in the box
positions = box_size * rand(N, 3);
% initialize velocities from the Maxwell-Boltzmann distribution
% YOUR CODE HERE #1

velocities=sqrt((kB*T)/mass).*randn(N,3);
% store positions every 10 steps. We will use this for visualization
save_interval = 10;
n_save = floor(n_steps / save_interval);
positions_history = zeros(n_save, N, 3);
```

```
    save_counter = 1;

    % store total momentum transfer
    total_momentum_transfer = 0;
    % Simulation loop
    for step = 1:n_steps
        % update positions
        % YOUR CODE HERE #2
        positions=positions+velocities*dt;

        % check particle collisions with the walls in each dimension
        for dim = 1:3
            % gas particles hitting the left wall at x=0 (or y=0, z=0)
            mask_left = (positions(:, dim) < 0);
            % reflect position if the particle has gone beyond the boundary
            positions(mask_left, dim) = -positions(mask_left, dim);
            % momentum transfer to the wall: 2*m*|v|
            total_momentum_transfer = total_momentum_transfer ...
                + 2 * mass * sum(abs(velocities(mask_left, dim)));
            % flip velocity after the particle is reflected
            velocities(mask_left, dim) = -velocities(mask_left, dim);
            % gas particles hitting the "right" wall at x=box_size (or
y=box_size, z=box_size)
            % YOUR CODE HERE #3
            mask_right = (positions(:, dim) > box_size);
            % reflect position
            % YOUR CODE HERE #4
            positions(mask_right, dim) = 2*box_size-positions(mask_right, dim);

            % momentum transfer
            % YOUR CODE HERE #5
            total_momentum_transfer = total_momentum_transfer ...
                + 2 * mass * sum(abs(velocities(mask_right, dim)));

            % flip velocity
            % YOUR CODE HERE #6
           velocities(mask_right, dim) = -velocities(mask_right, dim);
        end
        % save positions periodically
        if mod(step, save_interval) == 0
            positions_history(save_counter, :, :) = positions;
            save_counter = save_counter + 1;
        end
    end
    % calculate final pressure
    %    P = (Total momentum transfer) / (Total time * Wall area)
    simulation_time = n_steps * dt;
    pressure = total_momentum_transfer / (simulation_time * wall_area);
end
```

3) and 4)

```
function ideal_gas_sim
   %  Code to simulate MD for ideal gas molecules. Note the following:
   % - N non-interacting particles in a 3D cubic box.
   % - Collisions only with walls (particles bounce elastically).
   % - Pressure from total momentum transfer to walls.
   % - Compare simulated P vs. T to ideal gas law: P = (N/V) * kB * T.
   clear; clc; close all;
   %% Parameters
   N = 3;               % num of particles
   box_size_values = linspace(3,10,10) ;         % length of cubic box
   mass = 1.0;              % Particle mass
   kB = 1.0;                % Boltzmann constant ( we have set it to 1 here
for convenience)
   dt = 0.01;               % time step of the simulation
   n_steps = 1000;          % num. of simulation steps
   target_temperatures =1.0;  % Temp. range

   %% Allocate array to store pressure
    pressures = zeros(size(box_size_values));
    P_ideal = zeros(size(box_size_values));
   %% Loop over each temperature to perform MD at each temp
   for t_idx = 1:length(box_size_values)
       T = target_temperatures;
       box_size = box_size_values(t_idx);
       wall_area = 6 * box_size^2;  % Total surface area of the cubic box
       % run MD at this temp.
       [pressures(t_idx), positions_history] = simulate_ideal_gas( ...
           T, N, box_size, mass, n_steps, dt, kB, wall_area);
           %% Compare P-T data results from the sim w/ the ideal gas law: P =
(N * kB * T) / V
   % (kB=1 and V=box_size^3, therefore P_ideal = (N * T) / (box_size^3))
   volume = box_size^3;
   P_ideal(t_idx) = (N * T) / volume;
   end

   %% Plot results
   figure('Name','Pressure vs N_values','Color','w');
   plot(box_size_values, pressures, 'ro-','LineWidth',1.5,'MarkerSize',8);
   hold on;
   plot(box_size_values, P_ideal, 'bx-','LineWidth',1.5,'MarkerSize',8);
   xlabel('box_size_values');
   ylabel('Pressure (P)');
   title('Pressure vs box_size_values at T=1.0: Simulation vs Ideal Gas Law');
   legend('Simulated Pressure','Ideal Gas (PV = Nk_BT)','Location','best');
   grid on;
end
function [pressure, positions_history] = simulate_ideal_gas( ...
```

```
                T, N, box_size, mass, n_steps, dt, kB, wall_area)
    % IDEAL GAS SIMULATION
    % ---------------------------------------------------------
    % This function simulates an ideal gas in a box for n_steps at temperature
T
    % Returns:
    %   pressure
    %   positions_history

    % fix a random seed reproducibility
    rng(1);
    % initialize particle positions uniformly in the box
    positions = box_size * rand(N, 3);
    % initialize velocities from the Maxwell-Boltzmann distribution
    % YOUR CODE HERE #1

    velocities=sqrt((kB*T)/mass).*randn(N,3);
    % store positions every 10 steps. We will use this for visualization
    save_interval = 10;
    n_save = floor(n_steps / save_interval);
    positions_history = zeros(n_save, N, 3);
    save_counter = 1;

    % store total momentum transfer
    total_momentum_transfer = 0;
    % Simulation loop
    for step = 1:n_steps
        % update positions
        % YOUR CODE HERE #2
        positions=positions+velocities*dt;

        % check particle collisions with the walls in each dimension
        for dim = 1:3
            % gas particles hitting the left wall at x=0 (or y=0, z=0)
            mask_left = (positions(:, dim) < 0);
            % reflect position if the particle has gone beyond the boundary
            positions(mask_left, dim) = -positions(mask_left, dim);
            % momentum transfer to the wall: 2*m*|v|
            total_momentum_transfer = total_momentum_transfer ...
                + 2 * mass * sum(abs(velocities(mask_left, dim)));
            % flip velocity after the particle is reflected
            velocities(mask_left, dim) = -velocities(mask_left, dim);
            % gas particles hitting the "right" wall at x=box_size (or
y=box_size, z=box_size)
            % YOUR CODE HERE #3
            mask_right = (positions(:, dim) > box_size);
            % reflect position
            % YOUR CODE HERE #4
            positions(mask_right, dim) = 2*box_size-positions(mask_right, dim);
```

```
            % momentum transfer
            % YOUR CODE HERE #5
            total_momentum_transfer = total_momentum_transfer ...
                + 2 * mass * sum(abs(velocities(mask_right, dim)));

            % flip velocity
            % YOUR CODE HERE #6
           velocities(mask_right, dim) = -velocities(mask_right, dim);
        end
        % save positions periodically
        if mod(step, save_interval) == 0
            positions_history(save_counter, :, :) = positions;
            save_counter = save_counter + 1;
        end
    end
    % calculate final pressure
    %    P = (Total momentum transfer) / (Total time * Wall area)
    simulation_time = n_steps * dt;
    pressure = total_momentum_transfer / (simulation_time * wall_area);
end
```